Successful Development Efforts: Standards, People, & Culture: The Enterprise Perspective

Dr. Douglas C. Schmidt

Deputy Director, Research &
Chief Technology Officer

September 16, 2011

What's So Hard About Software Development?

Technical Complexities

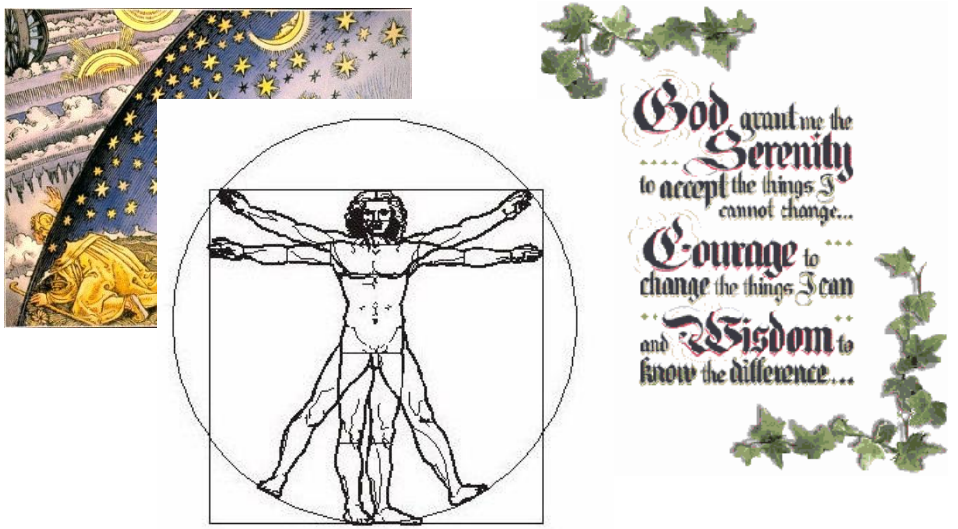
Accidental Complexities

- Low-level APIs & debugging tools
- Interoperability & portability

Inherent Complexities

- Quality of service (QoS) & security
- Scheduling & synchronization
- Intermittent connectivity
- …

Human Nature



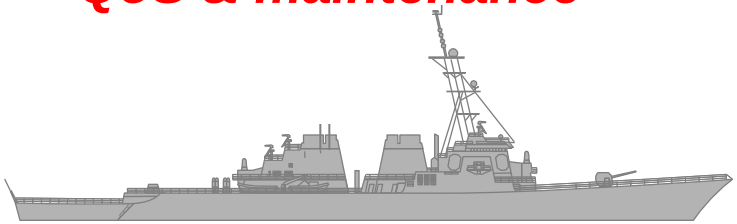
- Organizational impediments
- Economic impediments
- Policy impediments
- Political impediments
- Psychological impediments
- …

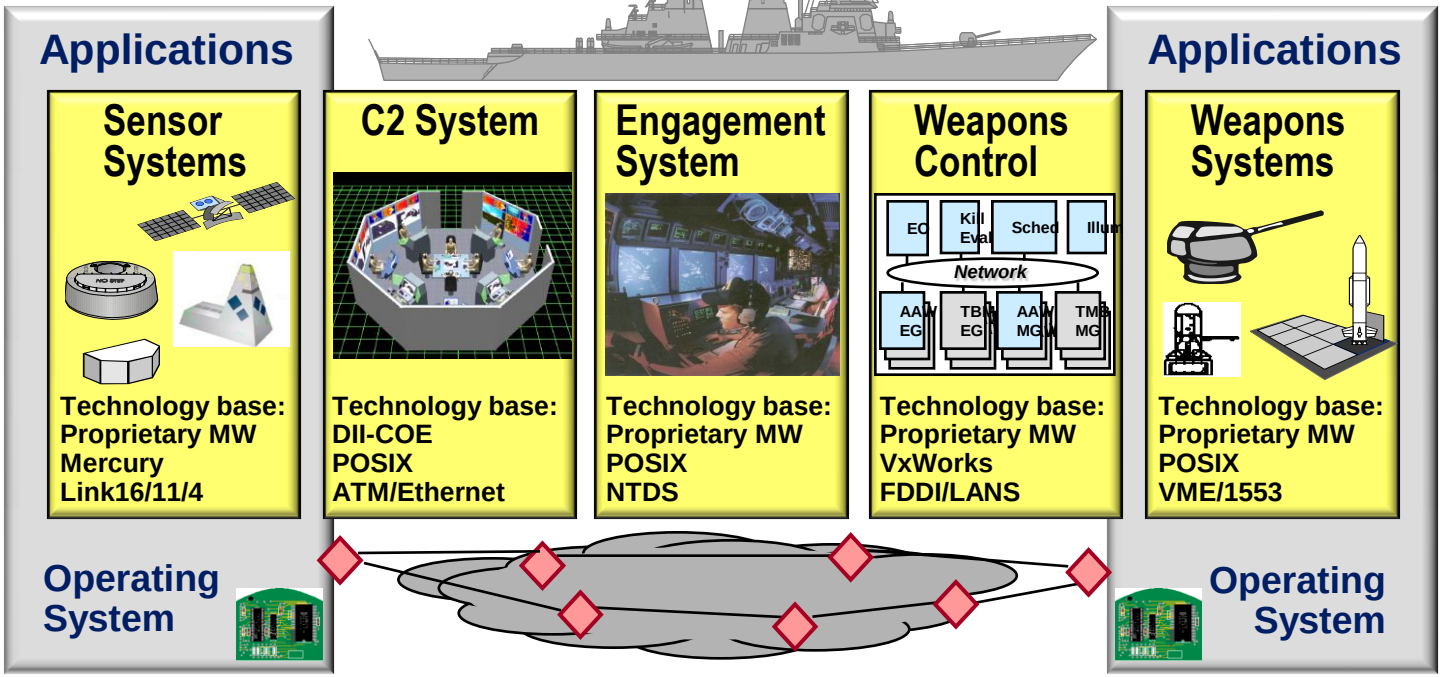
Evolution of DoD Software Development

Legacy DoD systems have historically been:

- Stovepiped
- Proprietary
- Brittle & non-adaptive
- Expensive
- Vulnerable

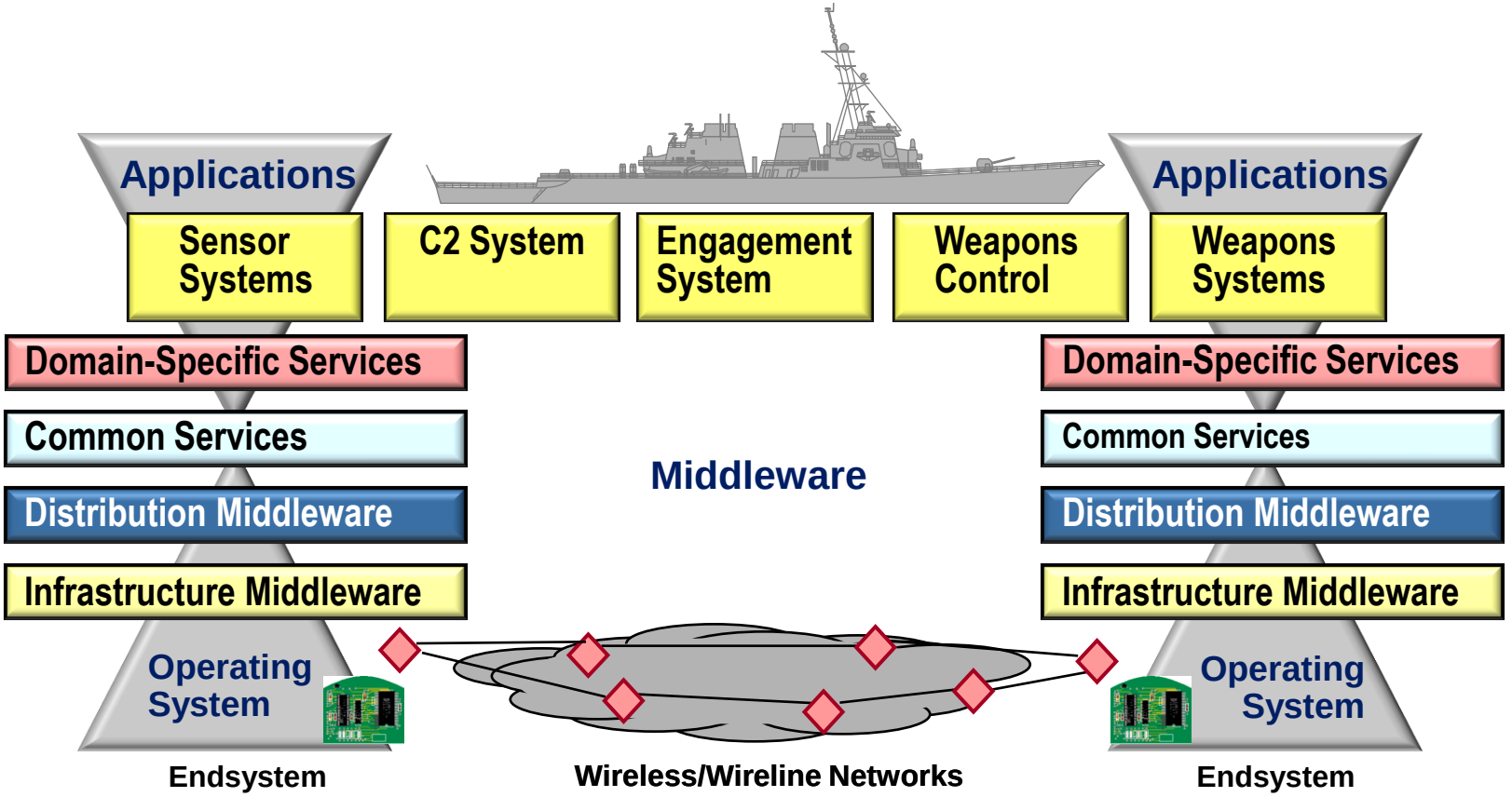
Consequence: Small HW/SW changes have big impact on DRE system QoS & maintenance



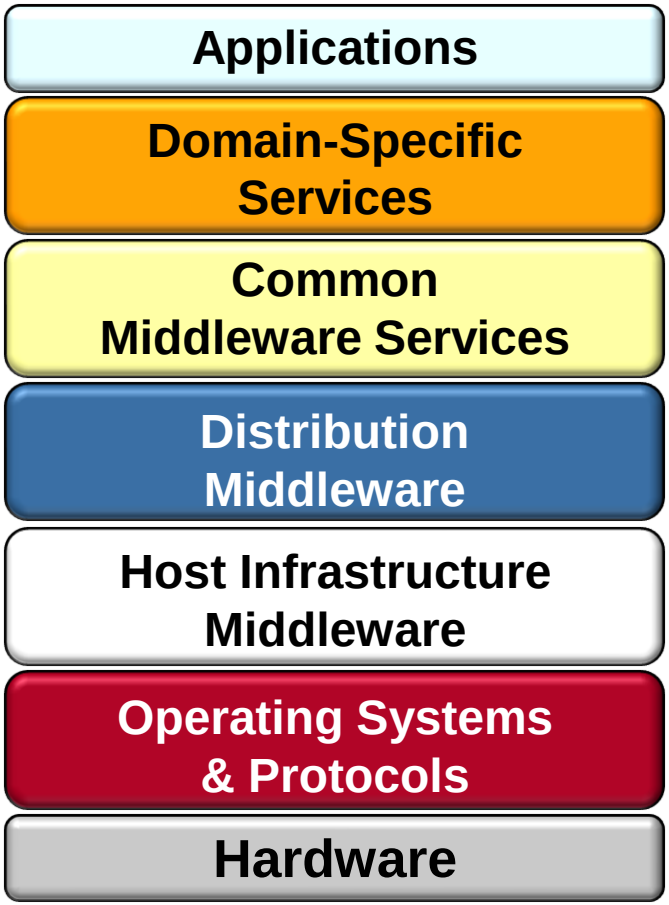


Evolution of DoD Software Development

- *Middleware* has effectively factored out many reusable services from traditional DRE application responsibility
  - Essential for product-line architectures, common operating environments, open architectures, etc.



The Evolution of Middleware



Historically, mission-critical apps were built directly atop hardware & OS

- Tedious, error-prone, & costly over lifecycles

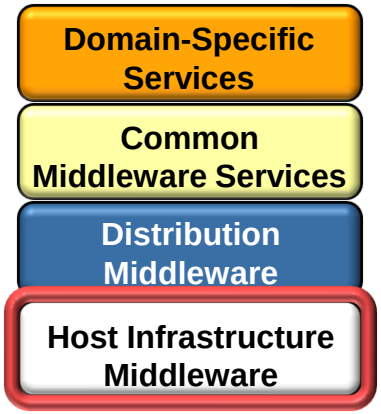
There are layers of middleware, just like there are layers of networking protocols

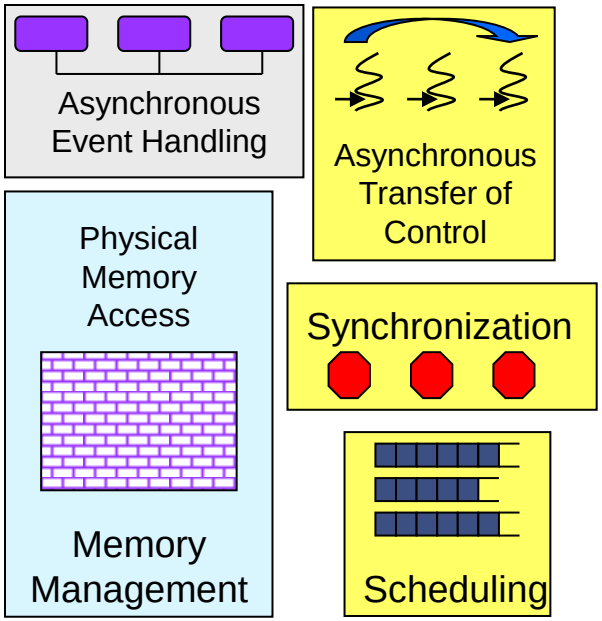
Standards-based COTS middleware helps support key mission goals:

- Control end-to-end resources & QoS
- Leverage hardware & software technology advances
- Evolve to new environments & requirements
- Provide a wide array of reusable, off-the-shelf developer-oriented services

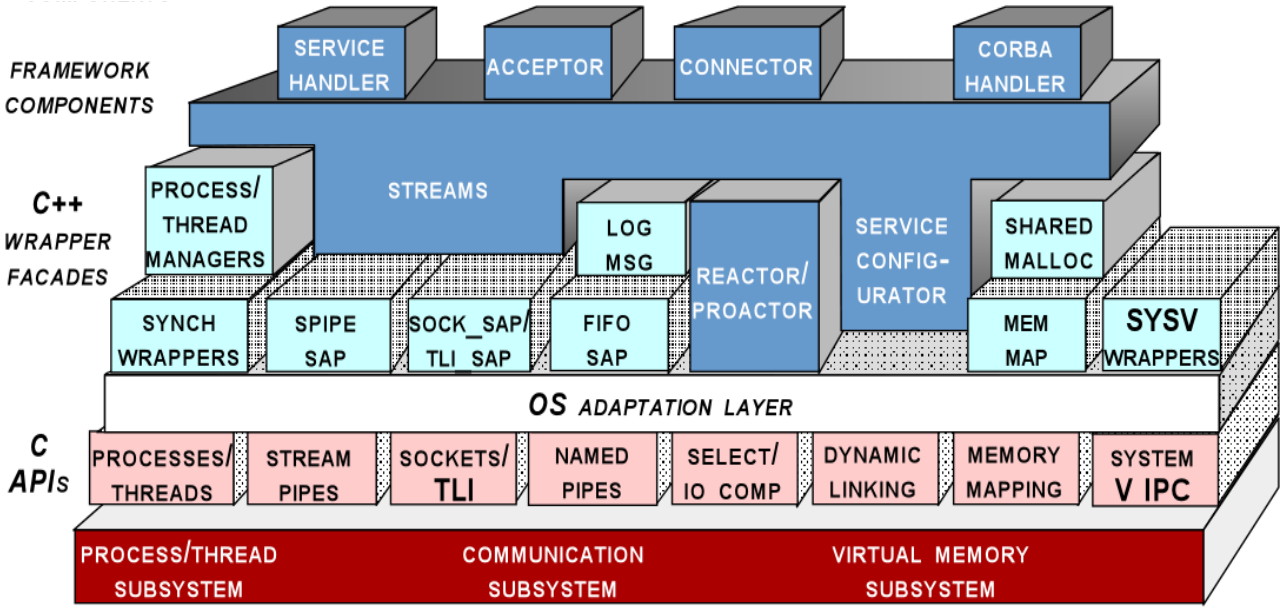
Host Infrastructure Middleware

- Host infrastructure middleware encapsulates & enhances native OS mechanisms to create reusable network programming components
- Examples
  - Java Virtual Machine (JVM), Common Language Runtime (CLR), ADAPTIVE Communication Environment (ACE)





www.rtj.org

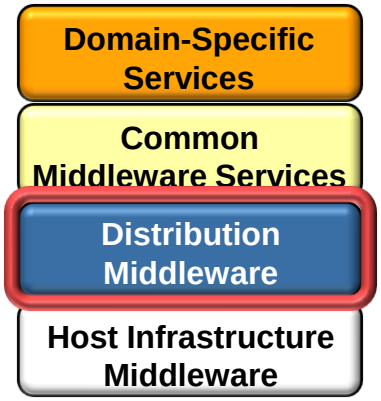


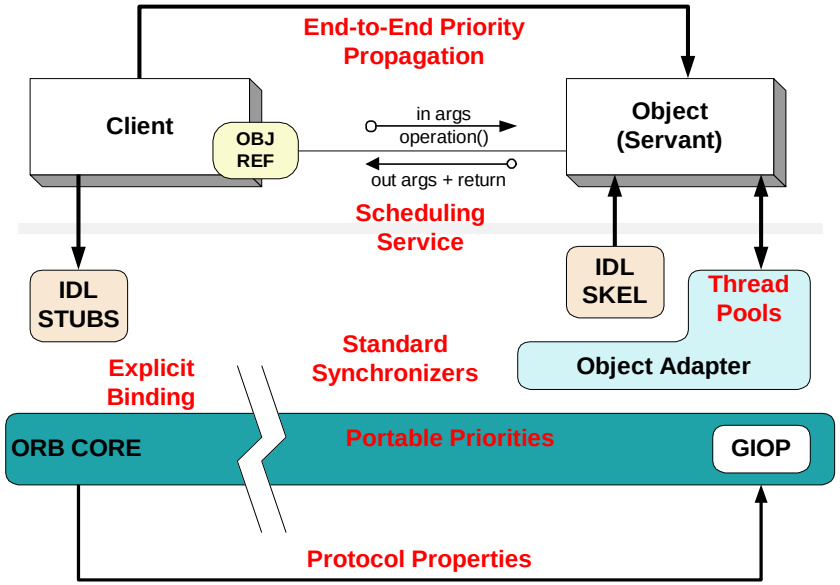
www.cs.wustl.edu/~schmidt/ACE.html

Host infrastructure middleware components abstract away many tedious & error-prone aspects of low-level OS APIs

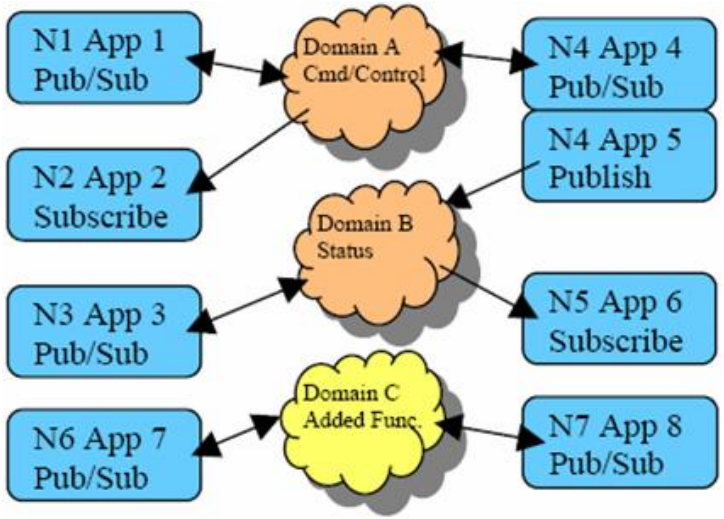
Distribution Middleware

- *Distribution middleware* defines higher-level distributed programming models whose reusable APIs & components automate & extend native OS capabilities
- Examples
  - OMG Real-time CORBA & the Data Distribution Service (DDS), W3C Simple Object Application Protocol (SOAP) Remote Procedure Calls (RPCs)





realtime.omg.org/

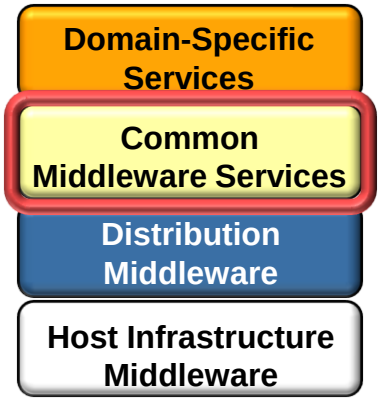


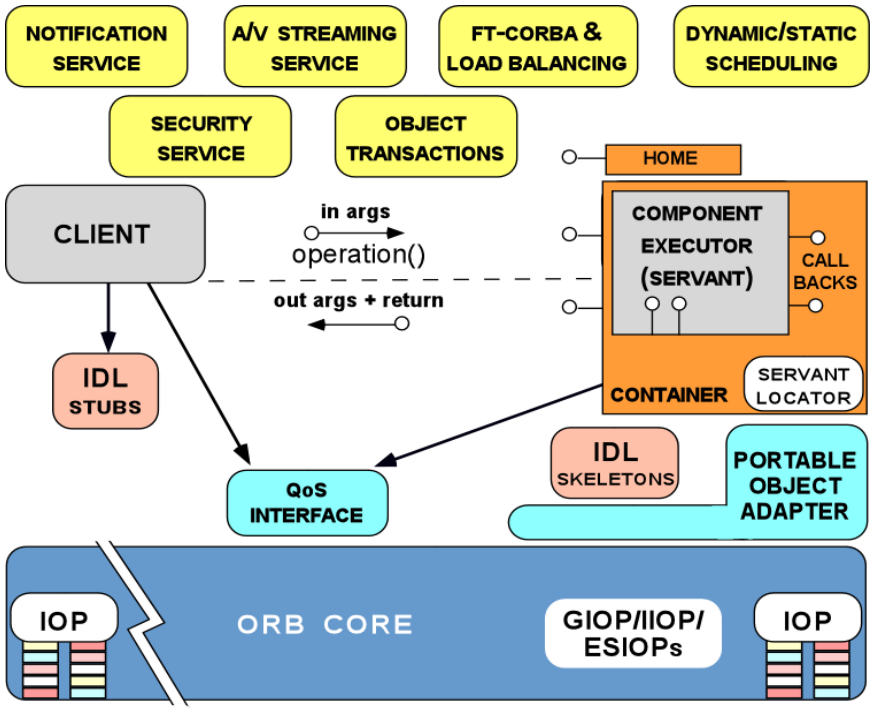
en.wikipedia.org/wiki/Data_Distribution_Service

Distribution middleware avoids hard-coding client & server application dependencies on object location, language, OS, protocols, & hardware

Common Middleware Services

- *Common middleware services* augment distribution middleware by defining higher-level domain-independent services that focus on programming "business logic"
- Examples
  - Sun's J2EE, Microsoft's .NET, W3C Web Services, CORBA Component Model & Object Services

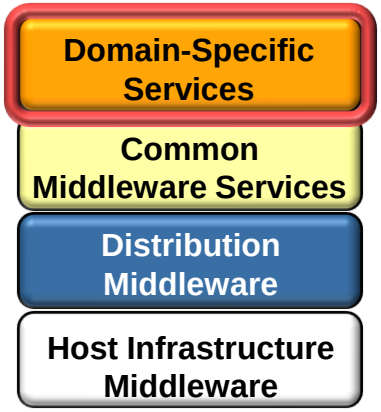


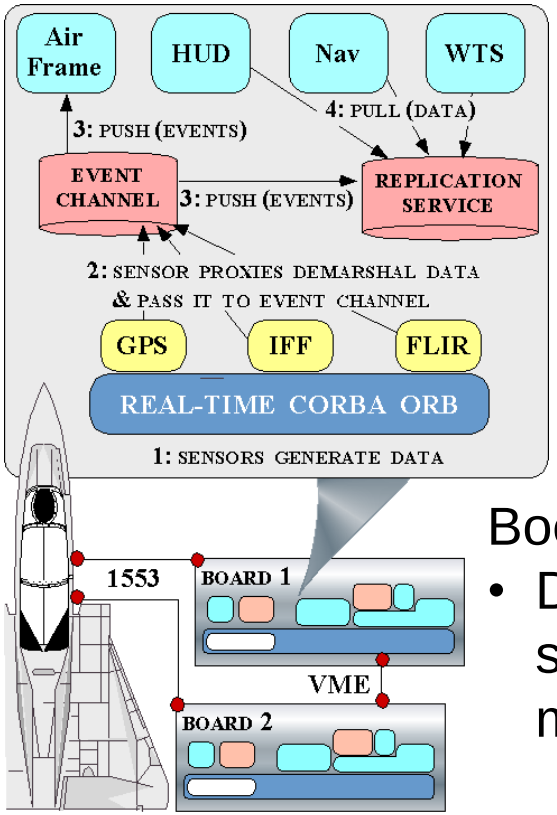


- Common middleware services support many recurring distributed system capabilities, e.g.,
  - Transactions & load balancing
  - Authentication & authorization
  - Database connection pooling & concurrency control
  - Active or passive replication
  - Dynamic resource management

Domain-Specific Middleware

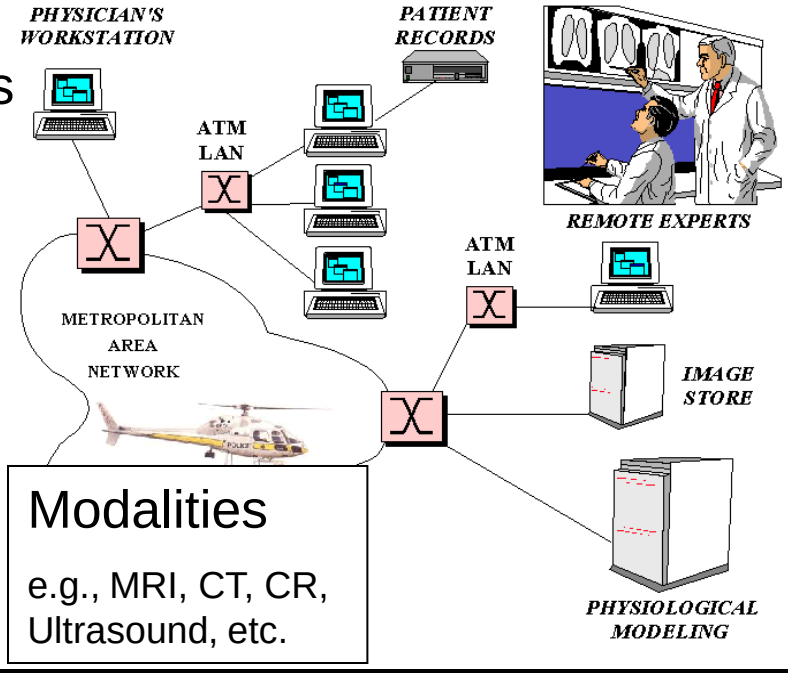
- *Domain-specific middleware services* are tailored to the requirements of particular domains, such as telecom, e-commerce, health care, process automation, avionics, etc.
- Examples





Siemens MED Syngo

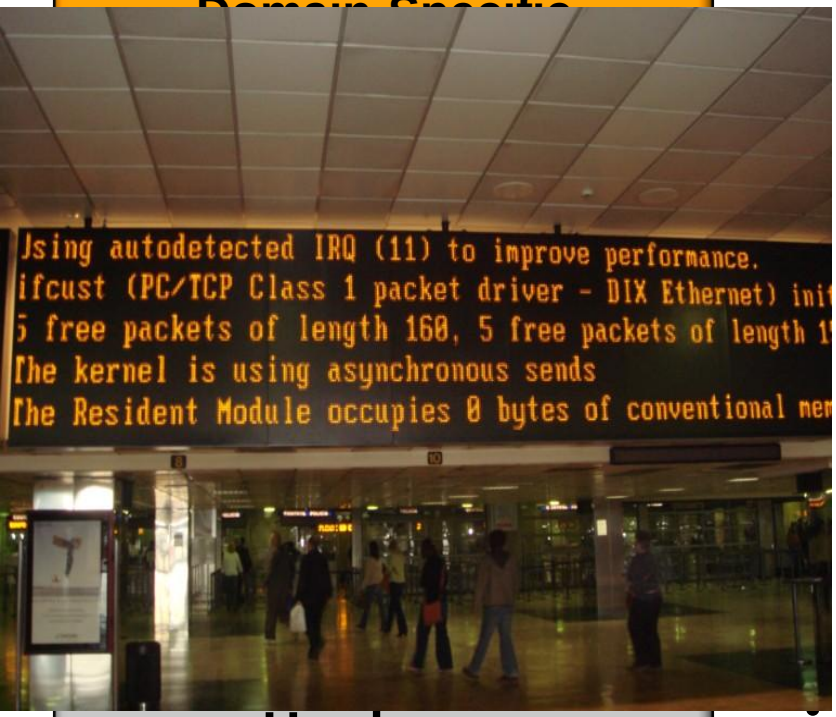
- Domain-specific services for distributed electronic medical systems
- Used by all Siemens MED business units worldwide



Boeing Bold Stroke

- Domain-specific services for avionics mission computers

Consequences of Standards & Commoditization



- More emphasis on integration rather than programming
- Increased technology convergence & standardization
- Mass market economies of scale for technology & personnel
- More disruptive technologies & global competition
- Lower priced—but often lower quality—hardware & software components
- The decline of internally funded R&D
- Potential for complexity cap in next-generation complex systems-of-systems

Not all trends bode well for traditional leaders

Ultimately, success depends on mastery of distributed real-time & embedded (DRE) systems

Ingredients for Software Development Success

Business Drivers

- i.e., need a “succeed or die” business case

Enlightened Managers

- Must be willing to defend the sacrifice of some short-term investment for long-term payoff

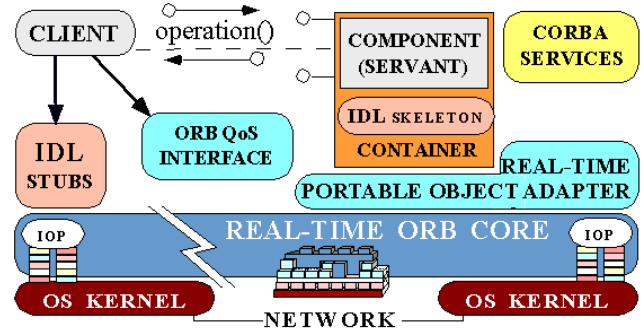
Experienced Senior Architects

- Responsible for communicating completeness, correctness, & consistency of the software architecture to stakeholders

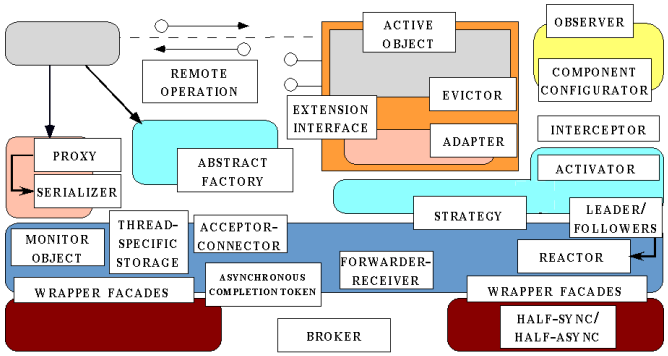
Solid Key Developers

- Responsible for design & evolution of specific architectural topic(s)/comps

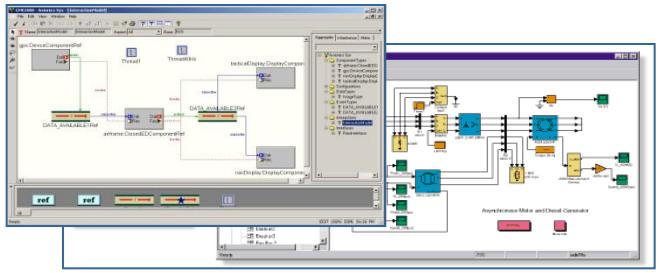
Key Technologies



Standard Middleware, Frameworks, Components, & Product Lines



Patterns & Pattern Languages



Model-driven Software Engineering

It’s crucial to have an effective process for growing architects & key developers

Traits of Dysfunctional Software Organizations

Process Traits

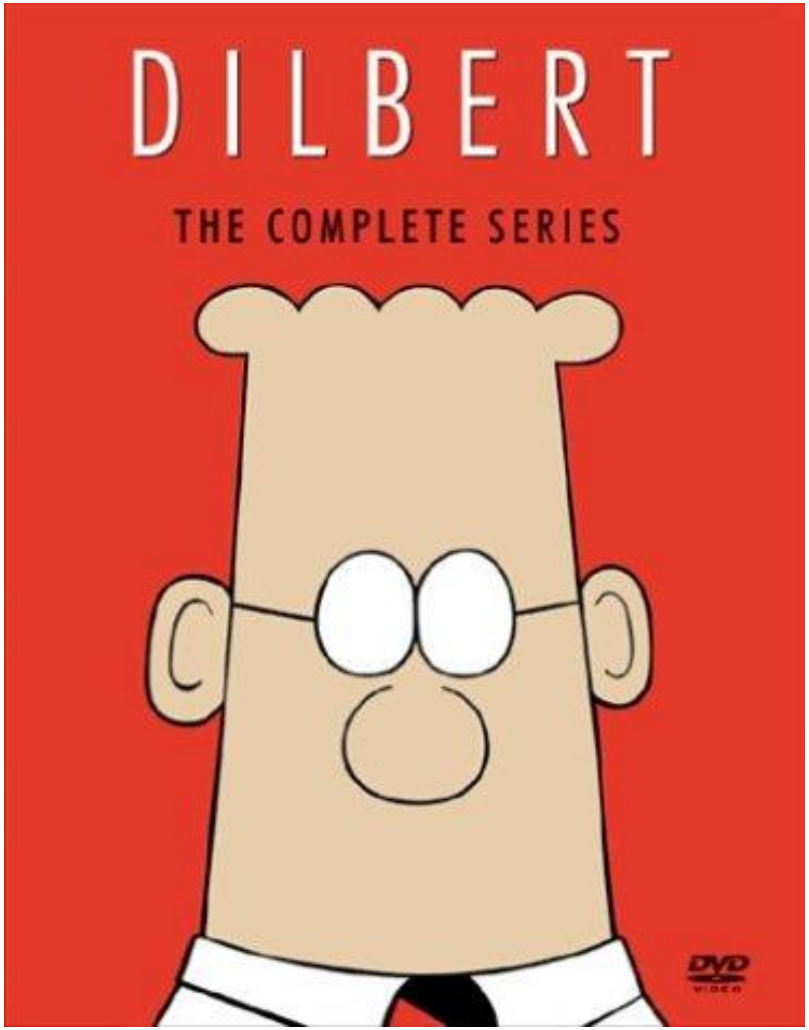
- Death through quality
  - "Process bureaucracy"
- Analysis paralysis
  - "Zero-lines of code seduction"
- Infrastructure churn
  - e.g., programming to low-level APIs

Organizational Traits

- Disrespect for quality developers
  - "Code monkeys" vs. "software artists"
- Top-heavy bureaucracy

Sociological Traits

- The "Not Invented Here" syndrome
- Modern method madness



www.cs.wustl.edu/~schmidt/editorials.html

Traits of Successful Software Organizations

Strong business & technology leaders

- Understand the role of software technology
- Don’t wait for “silver bullets”

Clear architectural vision

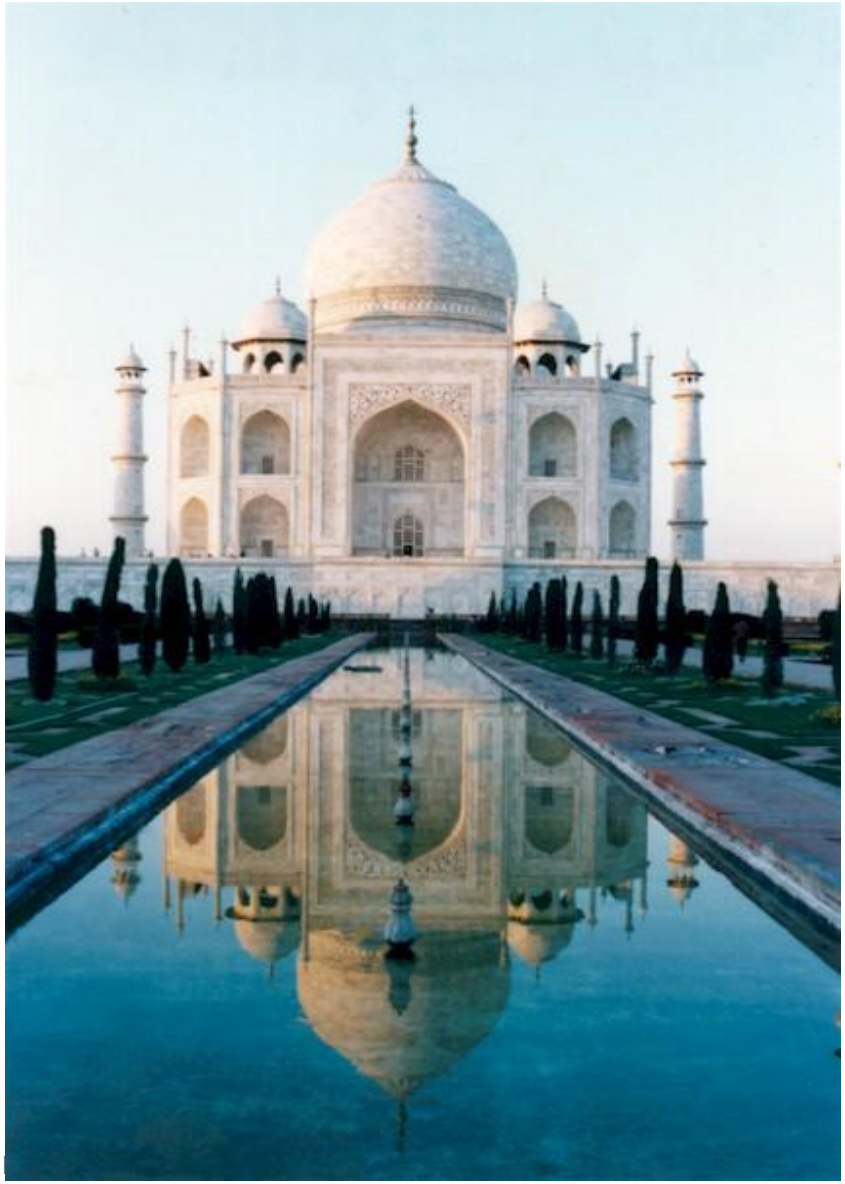
- Know when to build vs. buy
- Avoid worship of specific tools & technologies

Effective use of prototypes, demos, & spirals

- Reduce risk & get user feedback

Commitment to/from skilled developers

- Know how to motivate software developers & recognize the value of thoughtware



SEI

Concluding Remarks

Take-home Points

- Software-reliant systems often fail due to lack of
  - Awareness/acceptance of business drivers
  - Management commitment
  - Systematic mastery & application of key technologies
  - Developer education & training
- Success is achievable, though not easy
  - A good process is necessary, but *not* sufficient

False Prophets (Silver Bullets)

- Programming Languages
- Methodologies
- Processes
- Middleware
- Model-Driven Engineering
- Organization-central solutions
- Technology-centric solutions

THERE IS NO SUBSTITUTE FOR THINKING & HARD WORK!

See blog.sei.cmu.edu for more discussions of SEI software R&D activities